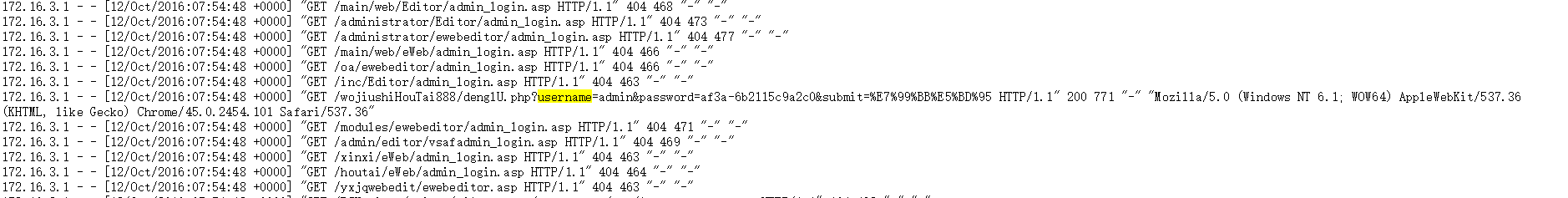
**Write up第二届上海市大学生网络安全大赛**

**Revengers**

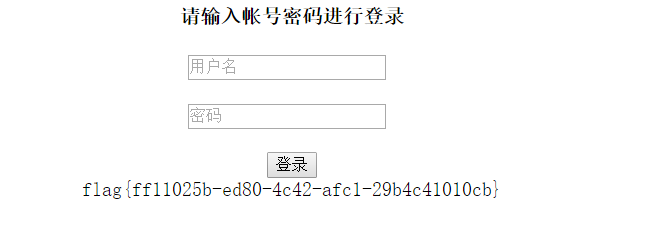
# WEB篇

## 0x01仔细

扫到后台有log路径，进入<http://120.132.85.112:15322/log/access.log>，找到一条非404的

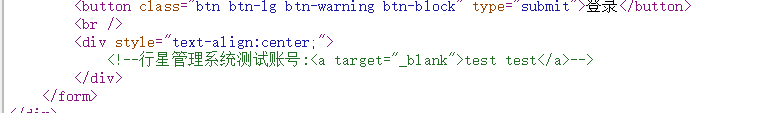


访问，即可找到flag

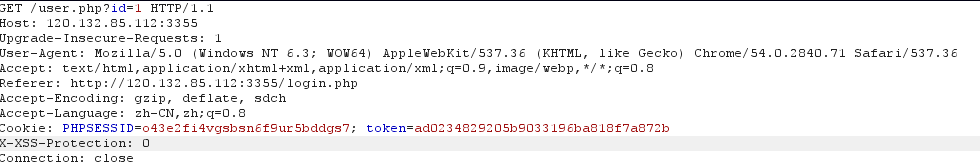


## 0x02跳

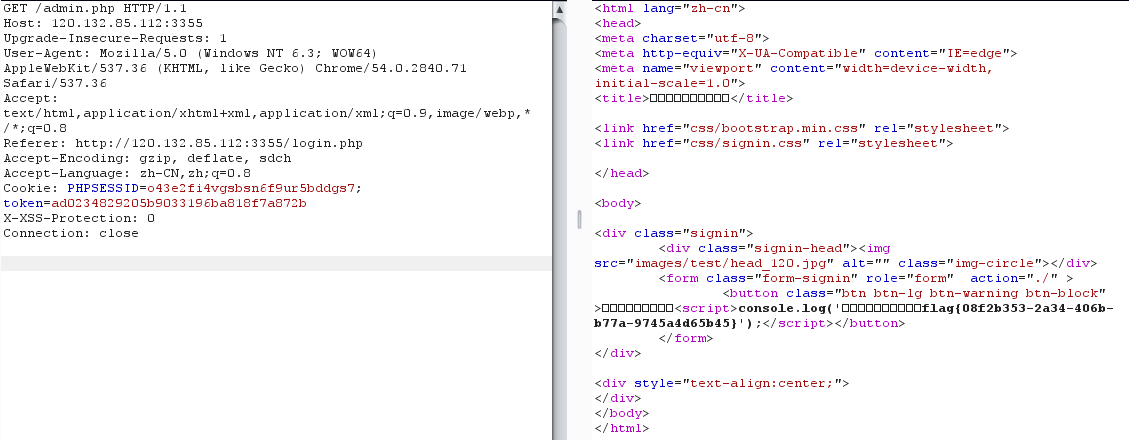
源码中给了测试账号test test。



登陆后找到cookie中的字段token：

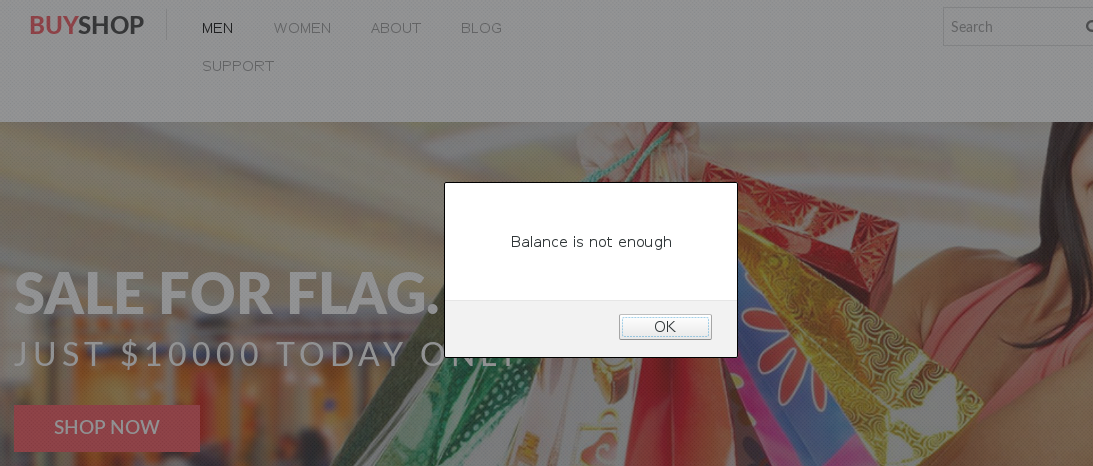


同时，扫描发现有admin.php的路径，输入admin.php会返回到login.php。尝试在访问admin.php时加入token，成功拿到flag。

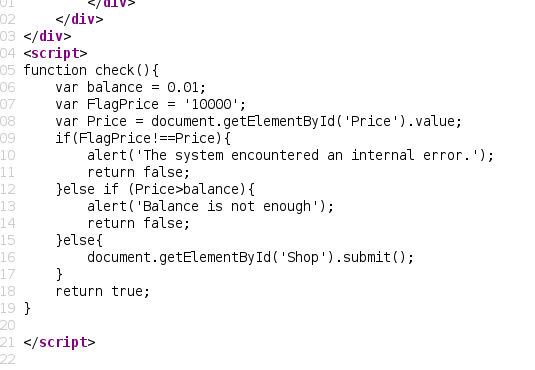


## 0x03物超所值

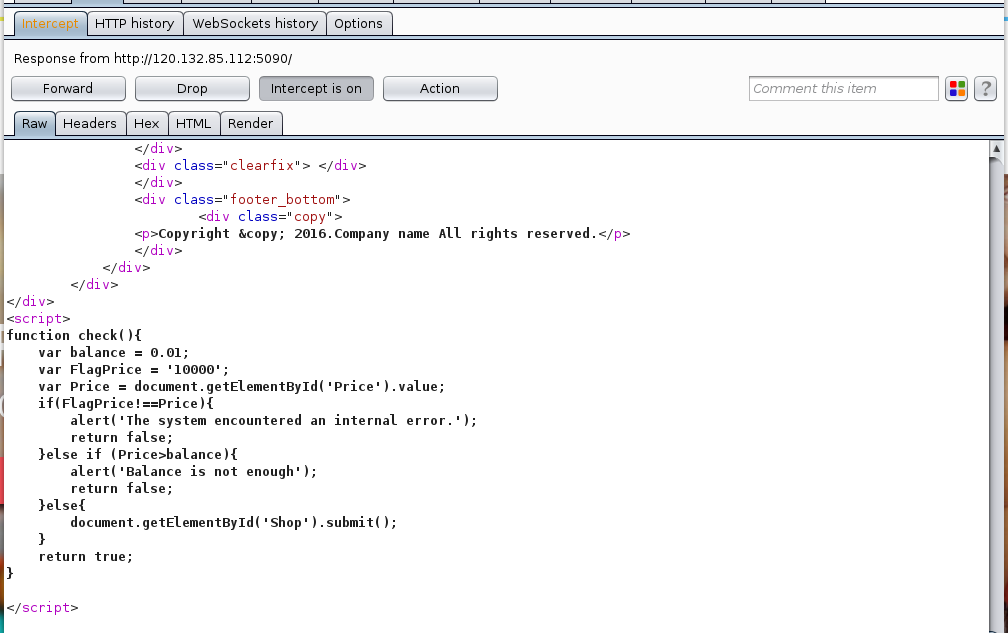
访问链接，点击SHOP NOW之后，弹出如下；这个过程中没有请求，所以是前台验证。



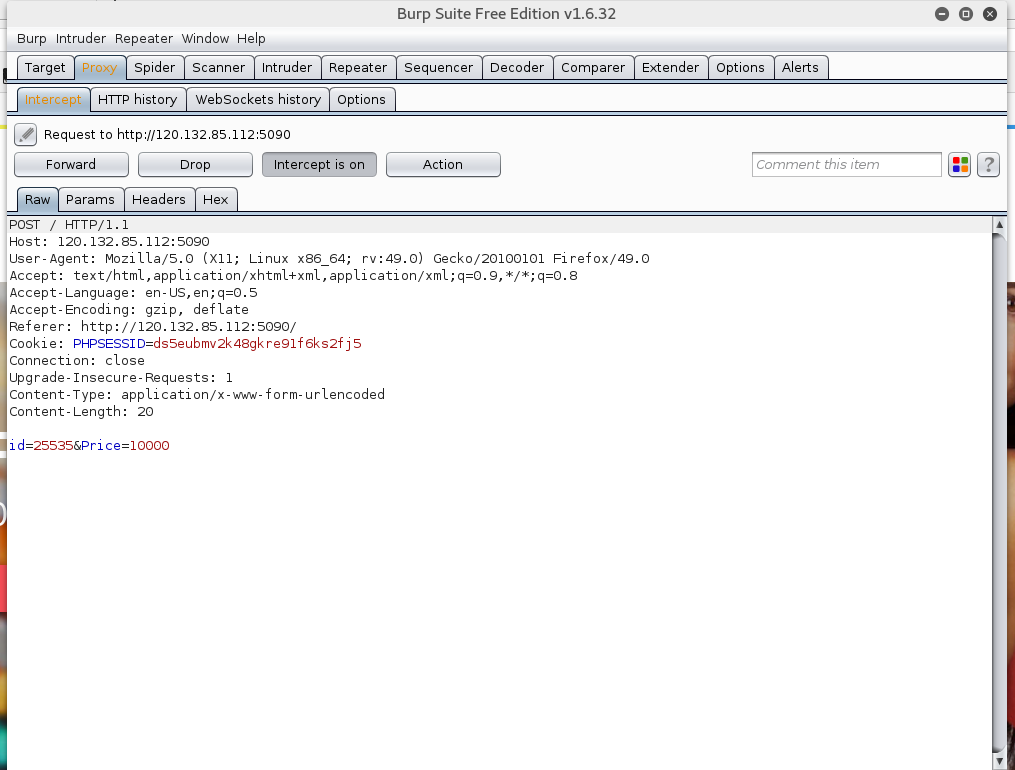
右键查看源代码，check函数，前台验证通过后，才向后台提交。

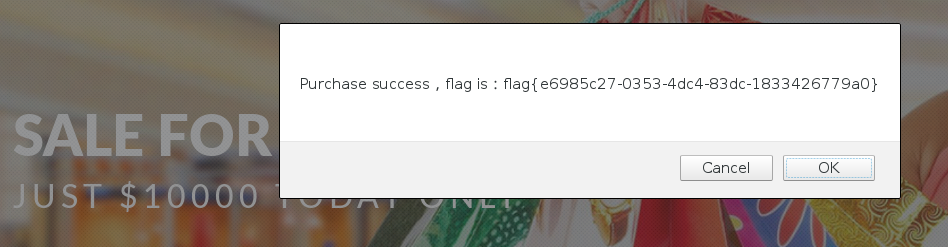


Burpsuite抓包，修改js中金额为10000，从而绕过前台判断。

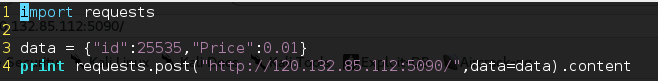


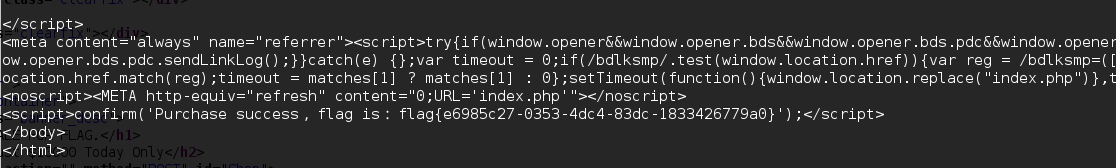
之后客户端post了如下数据，其中Price为flag的价格，推测后台拿Price与后台中你的balance比较，所以，修改Price为0.01即可。





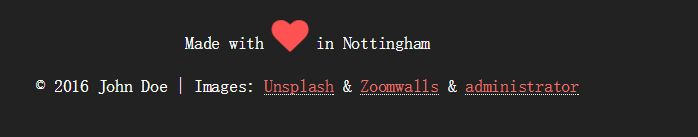
或者，更简单一点，可以直接向后台post数据。



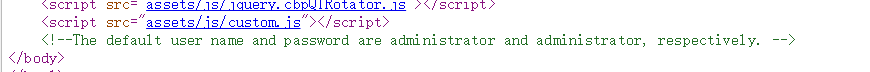


## 0x04分析

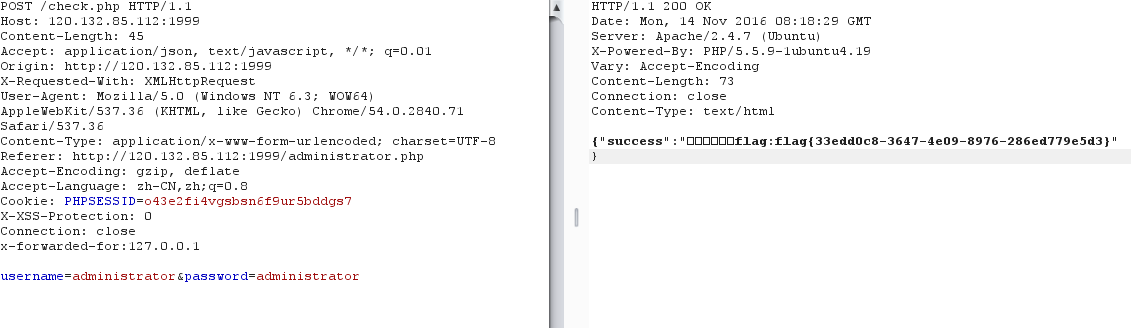
页面最下方有管理员登陆的链接：



在页面源码中找到账号密码…

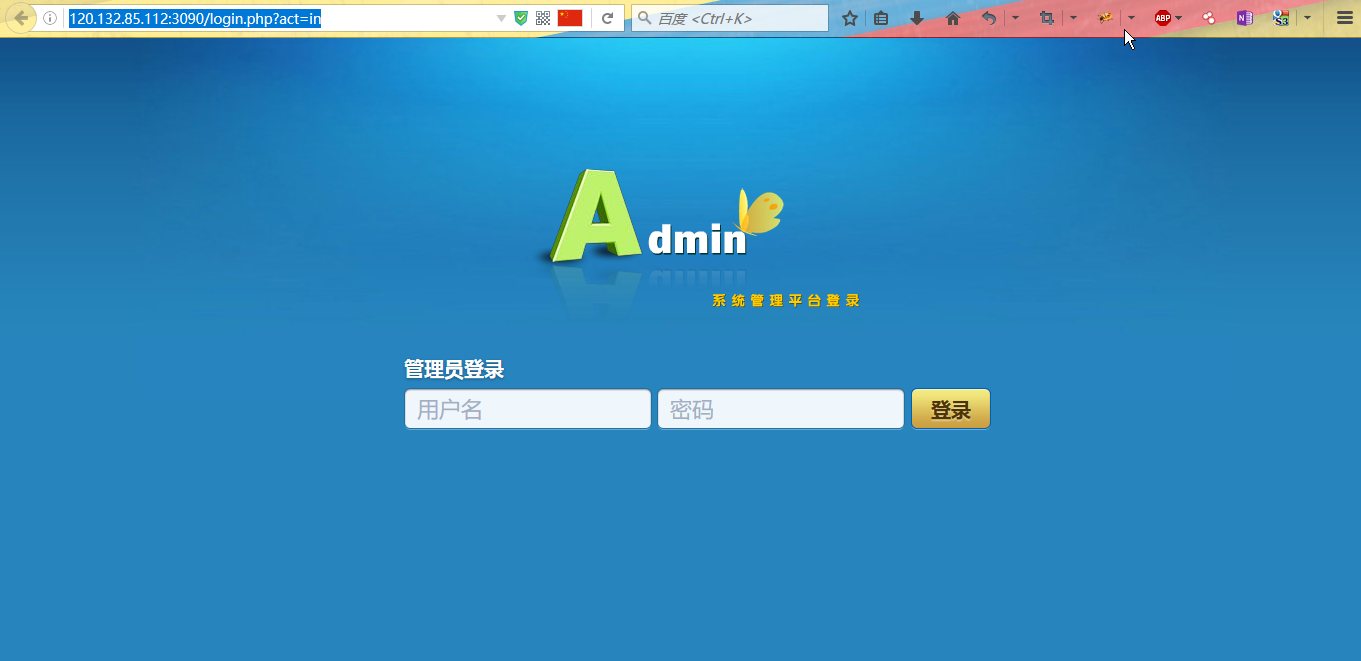


登陆提示IP不在许可范围内，修改x-forwarded-for:127.0.0.1。

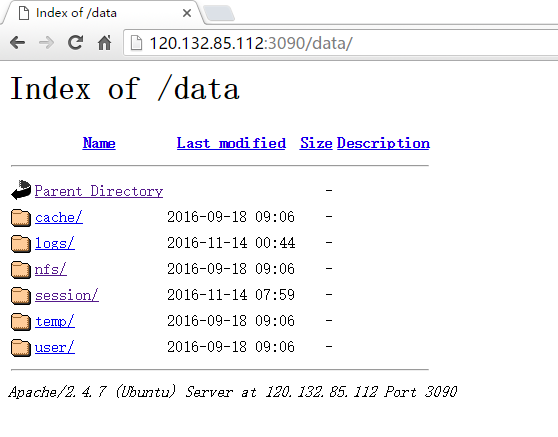


## 0x05威胁2

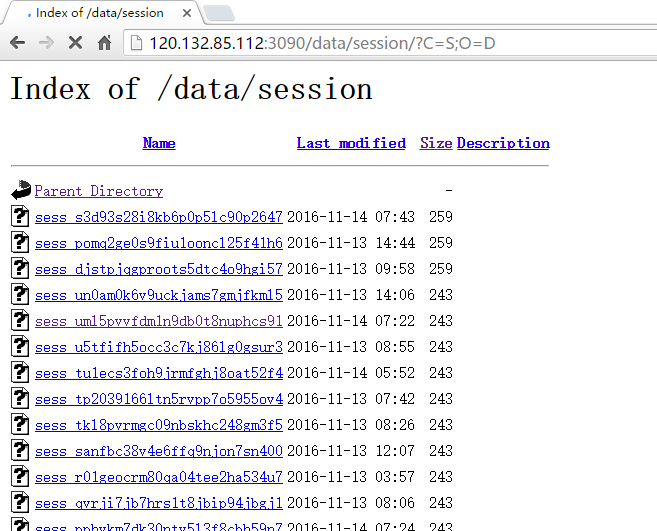
访问链接http://120.132.85.112:3090/login.php?act=in，页面如下：



扫描一下目录，发现存在data目录；



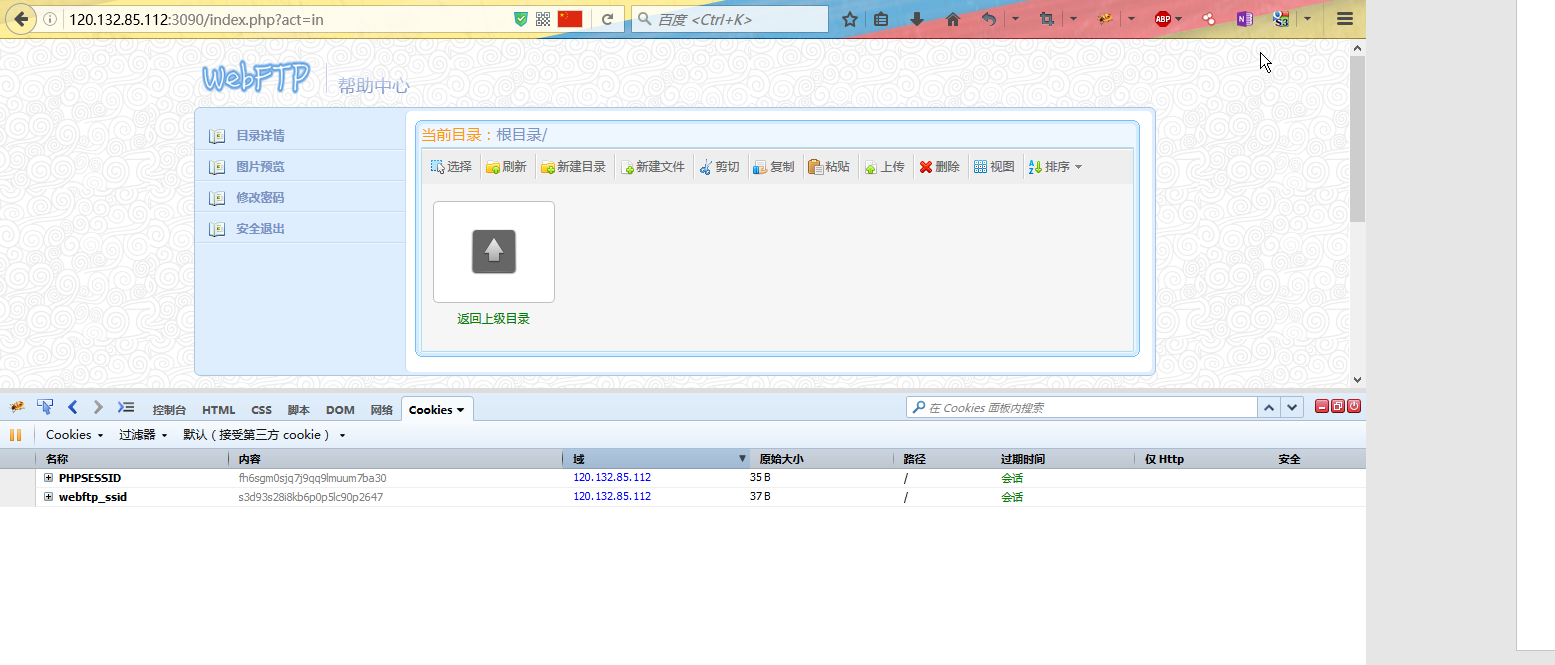
查看session目录【会非常非常卡因为有好多= =】，可以看到好多出现的session，然后我们按照size排序：



选了size最大的一个更换cookie里面的webftp\_ssid



然后访问index.php，顺利登陆到了系统里~



发现flag猥琐的藏在页面最下面

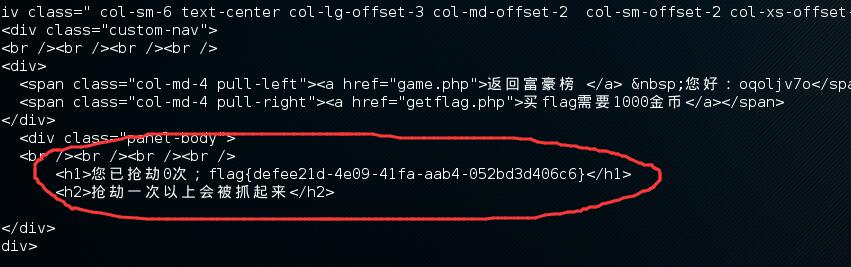


## 0x06抢金币

这道题以前出现过，可以抢别人的金币，但抢的时候得在五秒内输入验证码，就用python写了个识别验证码的脚本来抢金币。



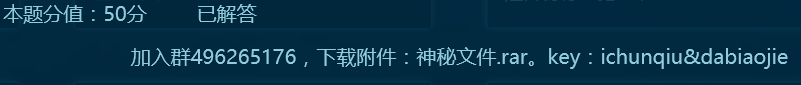
但这里题目作了更改，抢一次金币就会被抓住，这样即使抢到了1000金币也买不了flag。后来才发现发post数据抢金币会被抓，用get就不会….神脑洞….这是比赛之后才做出来的，比赛的时候抢金币网站崩了好久= =



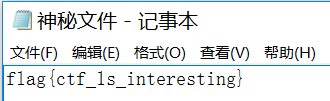
# MISC篇

## 0x01签到

根据提示加群后，下载文件，输入key解压压缩包。

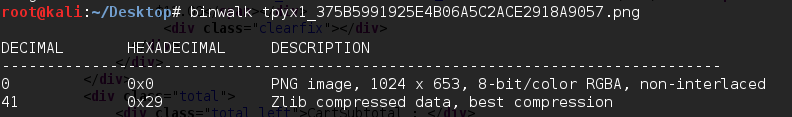


打开解压后的神秘文件，得到flag



## 0x02大可爱

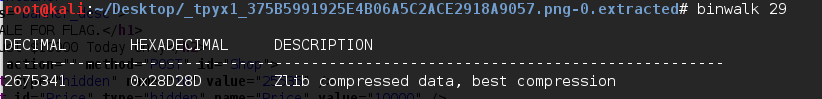
下载图片后，binwalk查看一下，图片中包含zlib压缩数据：



Binwalk -e提取后得到29和29.zlib，对29.zlib的数据解压后发现和29一样，那就从29入手了；



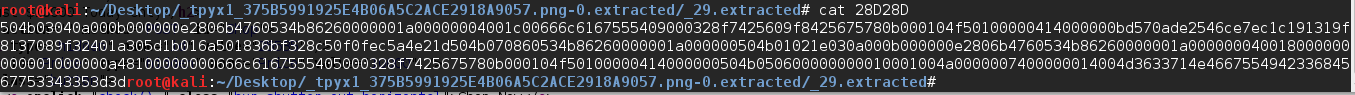
继续binwalk 29，原来里面还有一层zlib数据。Binwalk -e提取：



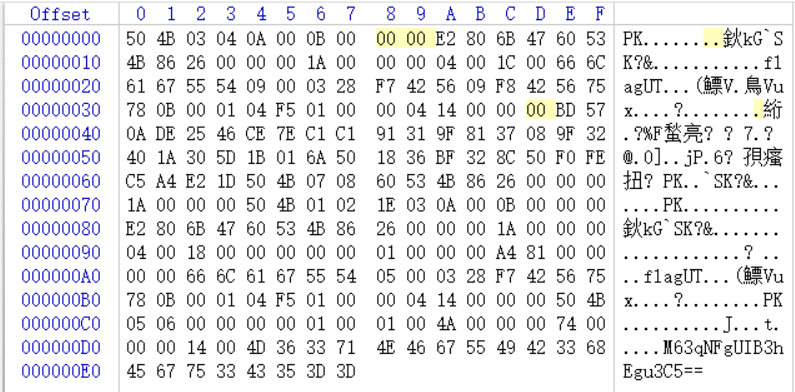
提取后得到这个，本能的以为这会是个无底洞；



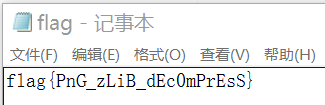
binwalk继续看28D28D，但这次什么都没有；直接cat 28D28D，看看文件内容。得到这一串，本以为这可能是什么加密，但恍惚间注意到开头是504b0304，明显的zip文件头：



因此把这段数据保存成了一个zip文件。

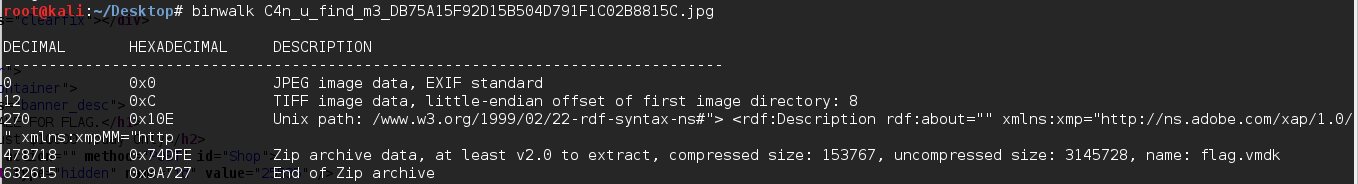


解压文件提示输入密码，上图中末尾那一串看起来是base64，可能是密码，base64解码后发现是乱码。无奈，直接把这串当做压缩包的密码试了一下，竟然还真是。解压后，得到flag



## 0x03面具

下载图片后，binwalk看一下，有如下发现：



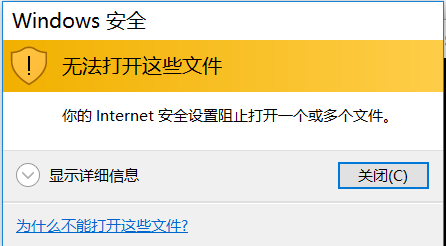
提取后得到一个flag.vmdk



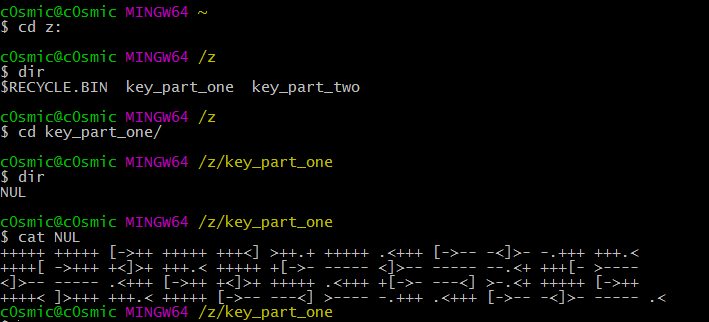
将其拷贝到windows下，右键映射为虚拟磁盘；可以看到里面有两个文件夹



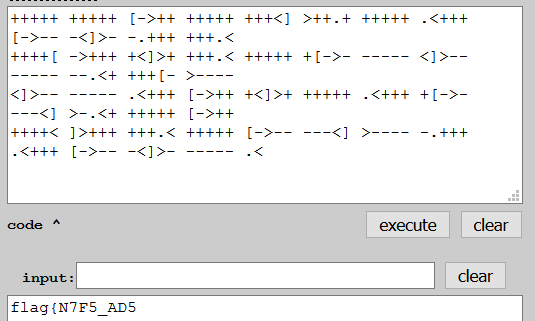
打开key\_part\_one，里面有个NUL文件，正常打不开。



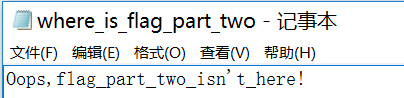
队友装了个git bash，可以在windows下执行Linux命令，读取NUL。



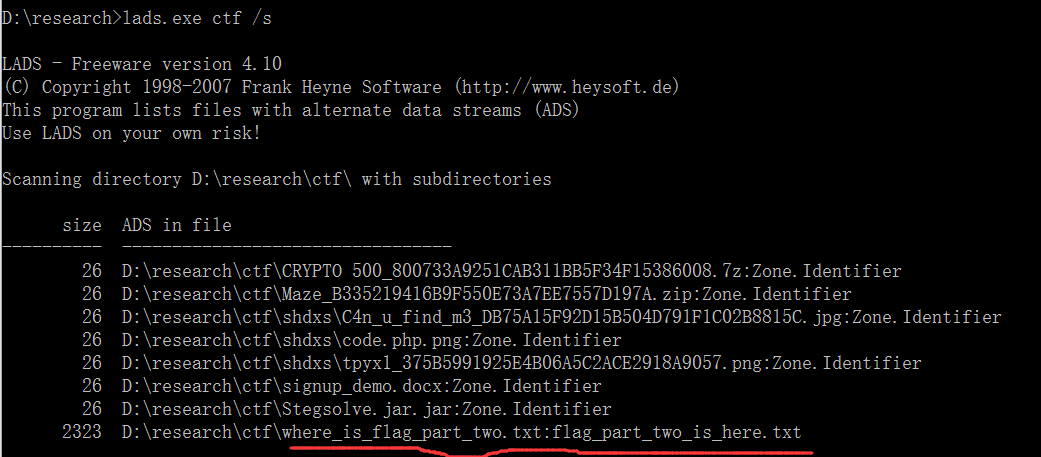
得到一串brainfuck，拿去运行，得到flag的第一部分。



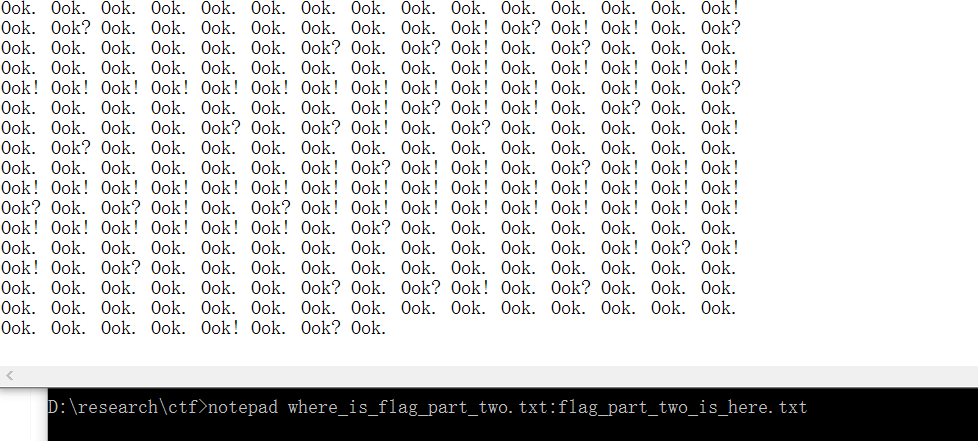
打开第二个文件夹，里面有个txt。



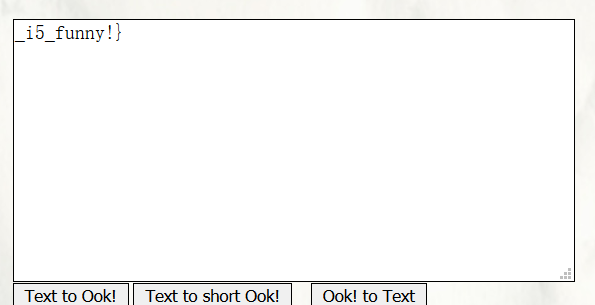
看一下文件大小，明显有隐藏数据，结合flag前半部分推测是ntfs隐藏。用lads查看一下，可以看到隐藏的文件是flag\_part\_two\_is\_here.txt



用notepad读一下，得到如下图所示的数据。这是什么东西？



队友说这也是brainfuck，<https://www.splitbrain.org/services/ook>，拿进去运行后，得到part\_two。

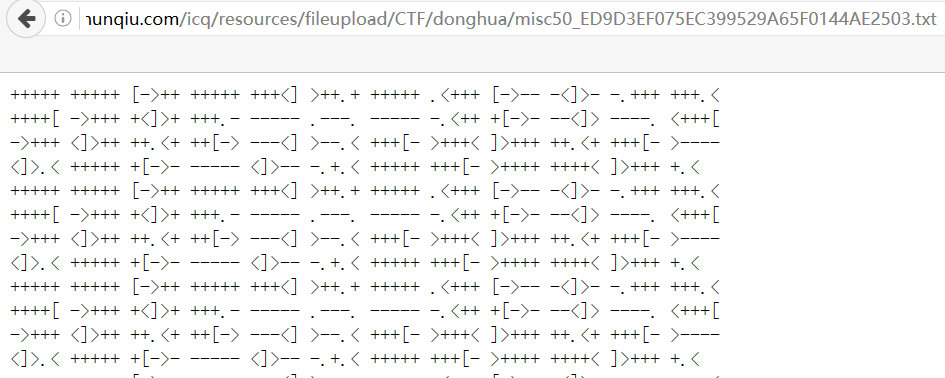


合起来就是flag{N7F5\_AD5\_i5\_funny!}

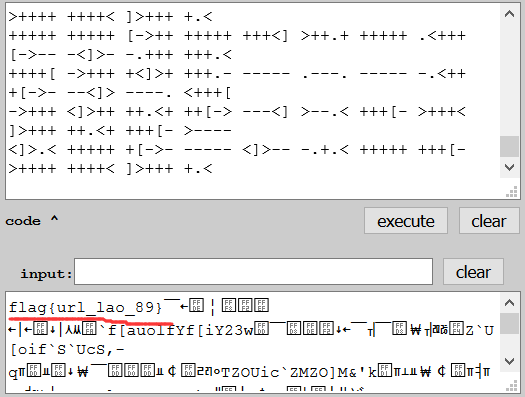
# CRYPTO篇

## 0x01 crypto50

附件打开后，是一段brainfuck代码



http://esoteric.sange.fi/brainfuck/impl/interp/i.html，执行即可得到flag



## 0x02 crypto300

洋葱…果真题如其名，得一层一层剥开

**1．第一关**

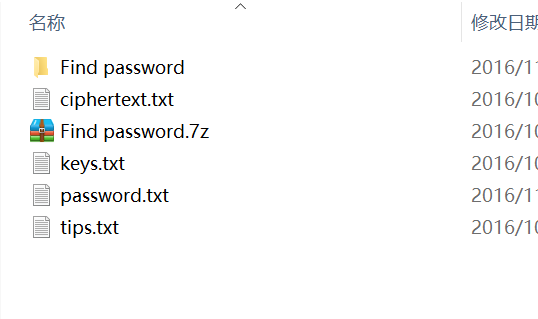
题目给的压缩包，很明显是考察crc32碰撞，每个是6字节，本来上脚本跑，但太慢了，上工具：https://github.com/theonlypwner/crc32



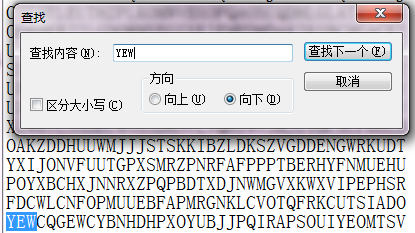
解之得到：\_CRC32\_i5\_n0t\_s4f3，作为解压缩密码来到第二关。

**2．第二关**

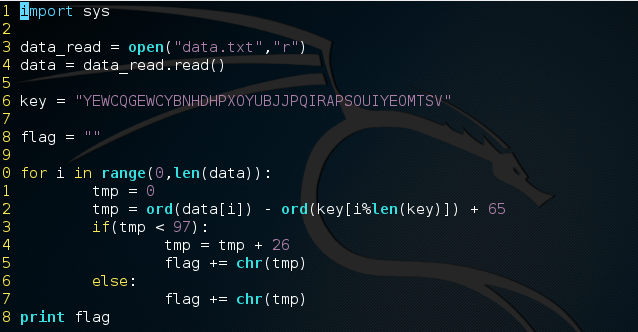
有如下文件

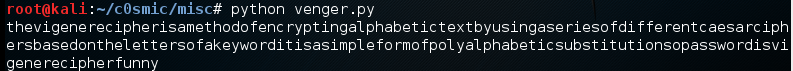


Tips.txt中提示维吉尼亚密码，且密钥在keys.txt中。打开密文ciphertext后，看到密文为rla xymi…..，想了一下三个字母的单词，常见的也就是“the”了，反向推算一下，那么密钥的前三个字母应该是YEW。于是，在keys.txt中搜索YEW，得到如下结果：



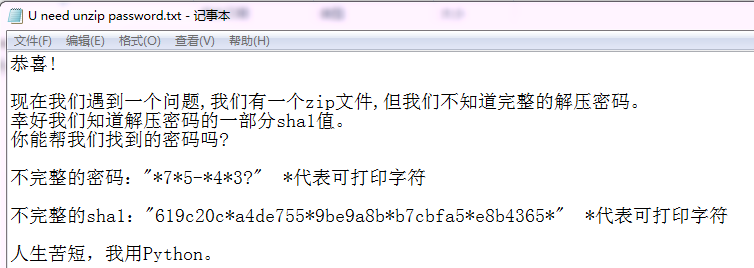
写了个脚本解密，得到明文，可以看到密码：vigenere cipher funny。





**3．第三关**

是一道sha1爆破题，题目如下：

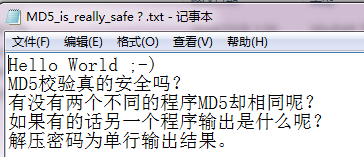


写脚本爆之即可，得到压缩包密码：I7~5-s4F3?

|  |
| --- |
| import string  import hashlib  import re  zidian = string.printable  pattern = re.compile(r'619c20c.a4de755.9be9a8b.b7cbfa5.e8b4365.')  print len(zidian)  count = 0  for i in zidian:  for j in zidian:  for m in zidian:  for n in zidian:  count = count + 1  tmp = i + "7" + j + "5" + "-" + m + "4" + n + "3?"  sha1\_value = hashlib.sha1(tmp).hexdigest()  if re.match(pattern,sha1\_value):  print tmp  if count%10000 == 0:  print count |

**4．第四关**

解压后，进入第四关，题目如下：



直接搜索关键词Hello World ;-) md5校验，可以看到这篇博客，

http://blog.csdn.net/liangkwok/article/details/7441867



程序下下来跑一跑，得到密码：Goodbye World :-(，解压文件进入第五关。

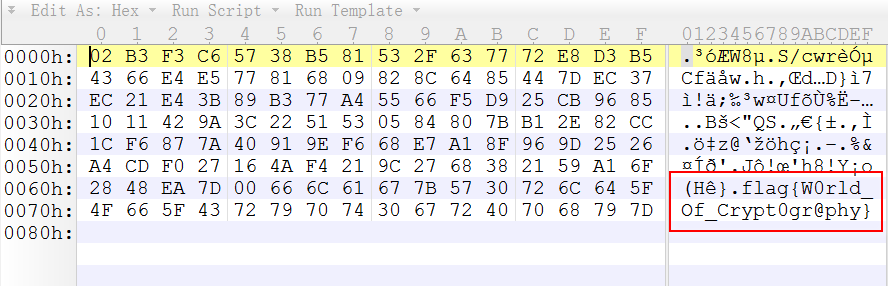
**5．第五关**

RSA算法，这道题由于n特别大，所以无法分解，但是又因为e和n很接近，所以维纳攻击，可以计算出d，工具链接：

<https://github.com/pablocelayes/rsa-wiener-attack>

脚本解密：

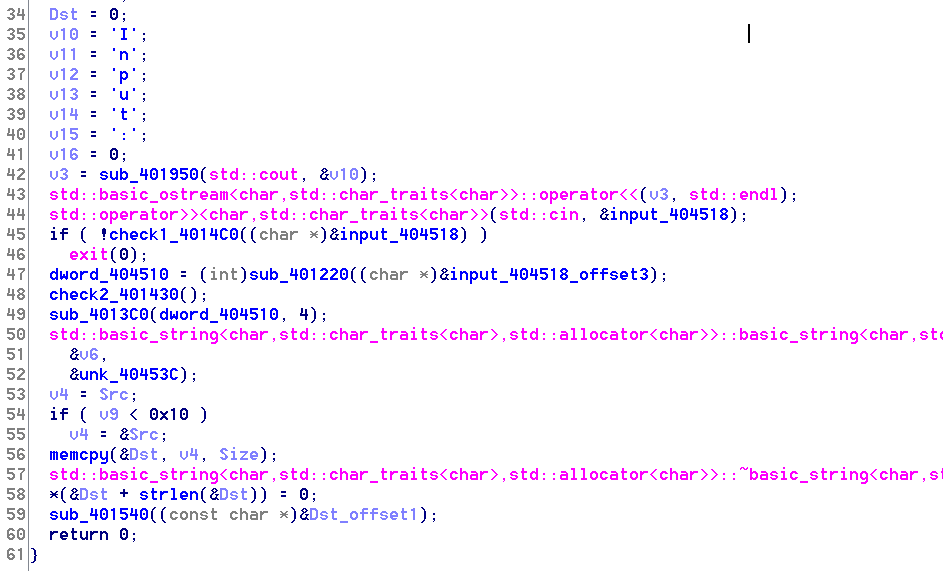
|  |
| --- |
| n=0x28FFF9DD3E6FE9781649EB7FE5E9303CF696347C4110BC4BA3969F0B11669840C51D81A6842B6DF2B090F21CD76D4371A8C0E47048C965ECA5B46913AFBB8DA052072A0566D7039C618ABA9065759B059E29E485DC5061A16AC63129438D9354E65DF5747546B85DB3D699819C4B7732DF927C7084A5D52D6E6D6AAC144623425  e=0x1f8fba410052df7eda3462f1aacd69e40760433ca335767cd7305a3d090805a5fd405dd6eea70e98f0ca1e1cf254748671bf0c98006c20eee1d6279043509fe7a98238b439160a5612da71e904514e81280617e307c3cd3313fa4c6fca33159d0441fbb18d83caf4bd46f6b9297a80a142dd69bf1a357ccb5e4c200b6d90f15a3  d=8264667972294275017293339772371783322168822149471976834221082393409363691895  with open("flag.enc","rb") as f:  data = f.read()  data = data.encode("hex")  c = int(data,16)  mingwen = pow(c,d,n)  print hex(mingwen)  result = hex(mingwen)[2:-1]  result = "0"+result  result = result.decode("hex")  with open("flag","wb") as f:  f.write(result) |



# REVERSE篇

## 0x01 maze

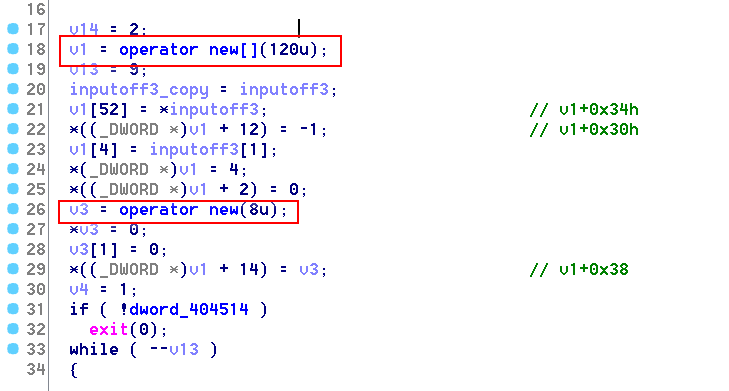
这道题比较复杂，尤其是对输入的check部分，分析了好久才明白怎么check的，做完后还是不理解这道题和迷宫有什么关系。用IDA打开，首先是main函数：



首先程序获取输入，并存储到input\_404518地址，check1\_4014c0是对输入做简单的check，要求输入的字符前三字节必须是410，并且所有的字符ASCII码值不能超过90，输入长度不大于25。

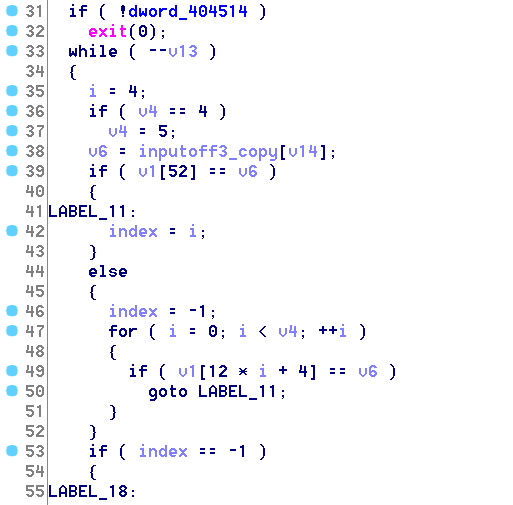
sub\_401220和check2\_401430分析起来就比较困难了，sub\_401220的参数是输入字符串的第四位置起，即"410"后面的内容。sub\_401220函数主要功能就是开辟了堆空间，并且用参数input\_404518\_offset3去初始化这个堆，这个堆其实上是程序自己定义的一个结构体数组，结构体如下：

|  |
| --- |
| struct Data{  int indexOfPreChar;  char[4] value;  struct AfterChar \* afterCharAddress;  };  struct AfterChar{  int index;  struct AfterChar \* afterChar;  }; |



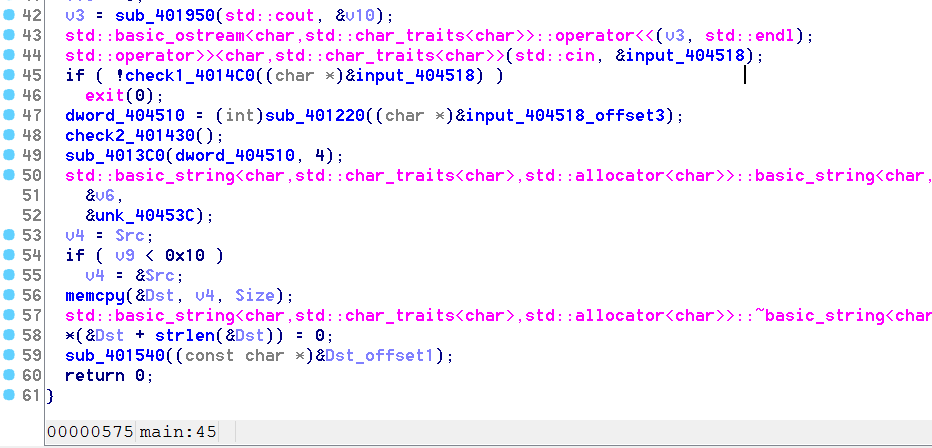
程序中长度为120的堆v1即是Data结构，后面开辟很多8字节的堆，就是AfterChar结构。Date结构的value[0]存储的是某个输入的字符，indexOfPreChar存储的是该字符第一次出现的时候前面一个字符在Data \* v1中的位置，afterCharAddress指的就是该字符的后面一个字符，比方字符'A'出现了三次，"ABACAD"，则afterCharAddress链表就按顺序存储了三个'A'后面的字符，即B、C、D，同样存储的是相应字符在Data \* v1中的位置。

sub\_401220所做的操作就是这样，以两个字母为一组，先把第一组的第一个字符存储在v1[4]中，再把第二个字符存储在v1[0]中，后面的每组字符的第一个跟v1中的字符比较，如果出现过，就把第二个字符加到v1中，并且相应的补充到该组第一个字符的AfterChar链表中，如果该组第一个字符没在v1出现过，则这组字符抛弃，继续下一组字符。



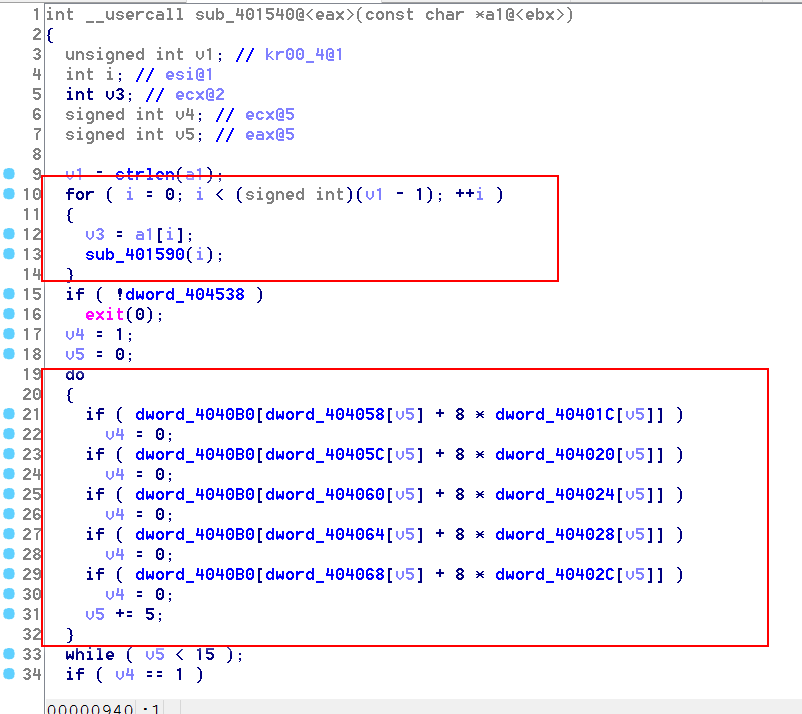
循环的v13初始值是9，所以可以判断参数的长度是8\*2+2=18字节，加上前面三字节"410"，输入长度为21字节。

check2\_401430即是对程序输入的第二个check，check条件大体是这样，不包括前面的"410"，即之后的18字节input\_404518\_offset3中，分为9组，input\_404518\_offset3[0]必须出现在3个组里并且都是第一个字符，input\_404518\_offset3[1]必须在两个组中位于第一个，input\_404518\_offset3[0]出现的最后一次的后一个字符，必须在三个组中位于第一个，即输入的格式是这样的：ABAEACBFBGCDDHDIDJ，由于sub\_401220在初始化堆表的条件，这里的"BF","BG","CD","DH","DI","DJ"六组顺序可以排列，只要保"CD"出现在"DH","DI","DJ"前就可以了。



sub\_4013c0函数的功能其实就是对输入input\_404518\_offset3去掉重复的字符，然后再通过后面字符串拷贝操作复制给Dst，所以Dst的长度为10。

sub\_401540传入的参数是&Dst[1]，主要有两个操作，一是对Dst[1]到Dst[8]作check，注意不包含Dst[0]的，check满足后将输入input\_404518\_offset3与内存中unk\_404094数组异或求出flag。



check部分由上述两块构成，第一处是对dword\_4040b0里面的内容8个为单位，进行移位，移位的位数由Dst[1]到Dst[8]分别减去48再模8决定，要获得flag的条件必须保证变更后的dword\_4040b0数组可以过掉第二处check，即满足下标为：

[3,11,12,13,14,22,28,29,30,36,43,44,50,51,58]

这15个位置必须为0。原先的dword\_4040b0：

01111111

00111100

11111110

01111100

11101111

11100111

11111100

01111111

能过第二处dword\_4040b0：

11101111

11100001

11111101

11110001

11110111

11100111

11001111

11011111

所以移位的位数为：3，5，7，6，1，4，0，2；所以可能的字符为：

x1 = ['3','C','K','S']

x2 = ['5','E','M','U']

X3 = ['7','G','O','W']

x4 = ['6','F','N','V']

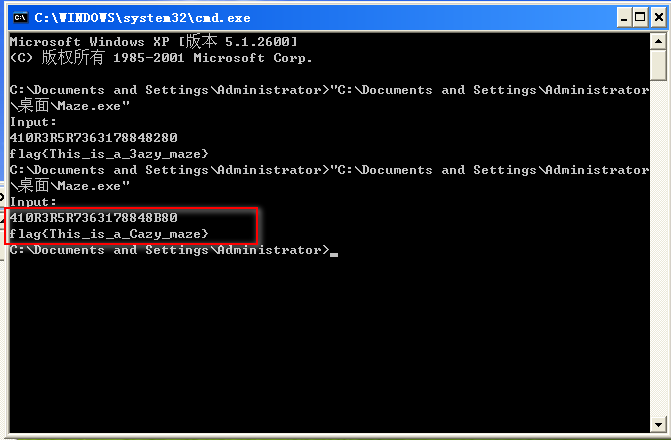
x5 = ['1','9','A','I','Q','Y']

x6 = ['0','8','H','P','X']

x7 = ['4','D','L','T']

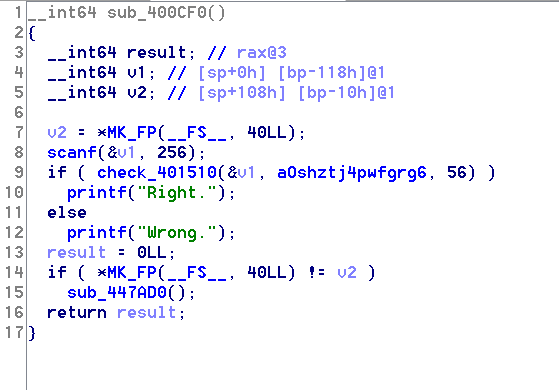
x8 = ['2','B','J','R','Z']

由于flag前5位为flag{，所以将其与unk\_404094数组前五位异或得到R3R5R，所以这就是为什么前面输入的格式是这样的了ABAEACBFBGCDDHDIDJ，后来通过排列组合加爆破，确定了input\_404518\_offset3的排列方式就是ABAEACBFBGCDDHDIDJ，心中万只草泥马奔过…早知就不写脚本，还亲自对爆破出的两万多个flag进行删选，泪奔…去掉重复的字符得到ABECFGDHIJ，并且字符的取值为R，3，5，7，6，1，8，4，B，0，即A:R，B：3，E：5，C：7，F：6，G：1，D：8，H：4，I：B，J：0。则程序输入为：410R3R5R7363178848B80。

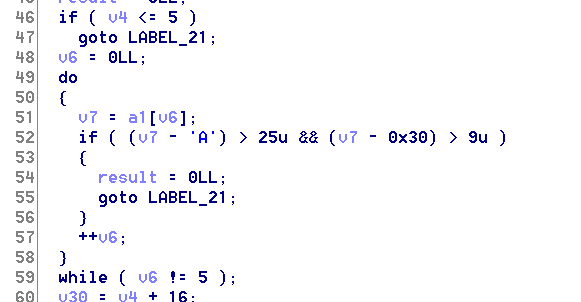


## 0x02 re400

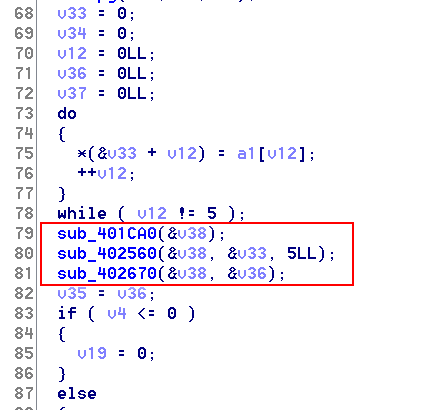
这道题看起来挺吓人的，那么多so库，又去掉了符号表，但仔细分析起来，还挺容易的，程序入口很简单：



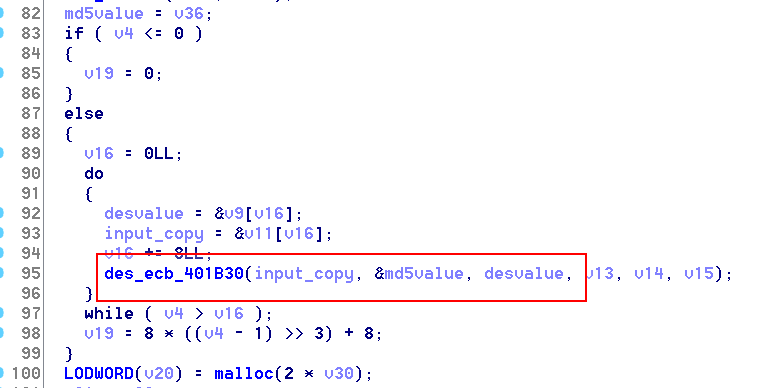
很明了，对输入做check，过了就输出right，错了就输出wrong，分析一下check\_401510这个函数。先是判断字符范围，ASCII码不能超过90。



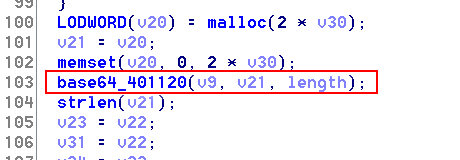
然后取输入的前5个字符，作md5加密。如下三个函数就是实现了md5算法。



接下来就是用前五个字符的md5值的前8字节作为DES算法的key，并且采用ECB模式分组加密。

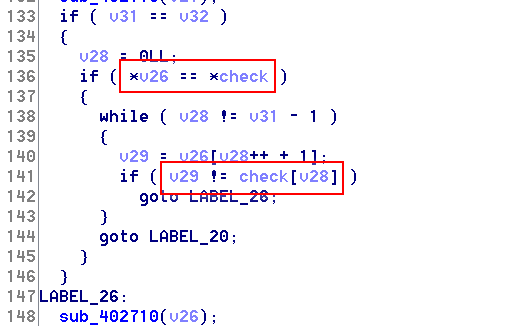


然后就是对DES加密后的结果作base64变换，v9是DES加密结果，v21保存base64结果。



最后就是和check数组比较，check数组内容即是：

'OSHzTJ4pwFgRG6eS6y3xVOOEGcbE5rzwqTs7VCK6ACQLuiTamZpXcQ=='



所以采用爆破的方式去得到flag，设flag前五位为temp，并且用其md5值前8字节作为DES算法的key，当解密出来的flag前五位和temp相等时，即为flag。

脚本如下：

|  |
| --- |
| import pyDes  import base64  import hashlib  import string  check = "OSHzTJ4pwFgRG6eS6y3xVOOEGcbE5rzwqTs7VCK6ACQLuiTamZpXcQ=="  miwen = base64.b64decode(check)  count = 0  for i in string.uppercase + string.digits:  for j in string.uppercase + string.digits:  for k in string.uppercase + string.digits:  for m in string.uppercase + string.digits:  for n in string.uppercase + string.digits:  count = count + 1  tmp = i+j+k+m+n  md5\_tmp = hashlib.md5(tmp).hexdigest()  key = md5\_tmp[0:16].decode("hex")  result = pyDes.des(key)  y = result.decrypt(miwen[:8])  if y[0:5] == tmp:  print y  if count % 10000 == 0:  print count |

解密结果：SHSECflag{675ac45bc131a1b7c145b605f4ba5}，感觉比maze那道题简单多了。